

Higher-order Processes with Parameterization over Names and Processes

Xian Xu *

East China University of Science and Technology, China

xuxian@ecust.edu.cn

Parameterization extends higher-order processes with the capability of abstraction and application (like those in lambda-calculus). This extension is strict, i.e., higher-order processes equipped with parameterization is computationally more powerful. This paper studies higher-order processes with two kinds of parameterization: one on names and the other on processes themselves. We present two results. One is that in presence of parameterization, higher-order processes can encode first-order (name-passing) processes in a quite neat fashion, in contrast to the fact that higher-order processes without parameterization cannot encode first-order processes at all. In the other result, we provide a simpler characterization of the (standard) context bisimulation for higher-order processes with parameterization, in terms of the normal bisimulation that stems from the well-known normal characterization for higher-order calculus. These two results demonstrate more essence of the parameterization method in the higher-order paradigm toward expressiveness and behavioural equivalence.

keywords: Parameterization, Context bisimulation, Higher-order, First-order, Processes

1 Introduction

In concurrent systems, higher-order means that processes communicate by means of process-passing (i.e., program-passing), whereas first-order means that processes communicate through name-passing (i.e., reference-passing). Parameterization originates from lambda-calculus (which is itself of higher-order nature), and enables processes, in a concurrent setting, to do abstraction and application in a way similar to that of lambda-calculus. Say P is a higher-order process, then an abstraction $\langle U \rangle P$ means abstracting the variable U in P to obtain somewhat a function (like $\lambda U.P$ in terms of lambda-calculus), and correspondingly an application $(\langle U \rangle P) \langle K \rangle$ means applying process K to the abstraction and obtaining an instantiation $P\{K/U\}$ (i.e., replacing each variable U in P with K , like $(\lambda U.P)K$ in terms of lambda-calculus). There are basically two kinds of parameterization: parameterization on names and parameterization on processes. In the former, U is a name variable and K is a concrete name. In the latter, U is a process variable and K is a concrete process. Parameterization is a natural way to extend the capacity of higher-order processes and this extension is strict, that is, the computational power strictly increases with the help of parameterization [12]. In this paper, we study higher-order processes in presence of parameterization.

Comparison between higher-order and first-order processes is a frequent topic in concurrency theory. Such comparison, for example, asks whether higher-order processes can correctly express first-order processes, or vice versa. It is well known that first-order processes can elegantly encode higher-order processes [19, 22]; the converse is however not quite the case. As the first issue, this paper addresses how to encode first-order processes with higher-order processes (equipped with parameterization).

*This work has been supported by project ANR 12IS02001 PACE and NSF of China (61261130589, 61472239, 61572318).

The very early work on using higher-order process to interpret first-order ones is contributed by Thomsen [24], who proposed a prototype encoding of first-order processes with higher-order processes with the relabelling operator (like that in CCS [16]). This encoding uses a gadget called wire to mimic the function of a name in the higher-order setting, and essentially employs the relabelling to make the wires work properly so as to fulfill the role of names. Due to the arbitrary ability of changing names (e.g., from global to local), the encoding has a correct operational correspondence (i.e., the correspondence between the processes before and after the encoding), but is very hard to analyze for full abstraction (i.e., the first-order processes are equivalent if and only if their encodings are; the ‘if only’ direction is called soundness and the other direction is called completeness). Unfortunately, without the relabelling operator, the basic higher-order process (which has the elementary operators including input, output, parallel composition and restriction) is not capable of encoding first-order processes [25]. In the literature, several variants of higher-order processes are exploited to encode first-order processes. In [22], an asynchronous higher-order calculus with parameterization on names is used to compile the asynchronous localized π -calculus (a variant of the first-order π -calculus [17]). This encoding depends heavily on the notions of ‘localized’ which means only the output capability of a name can be communicated during interactions, and ‘asynchronous’ which means the output is non-blocking. Though technically a nice reference, intuitively because this variant of π -calculus is less expressive than the full π -calculus, it is not very surprising that the higher-order processes with parameterization on names can interpret it faithfully, i.e., fully abstract with respect to barbed congruence. Then in [28], we explore the encoding of the full π -calculus using higher-order processes with parameterization on names. In that effort, we construct an encoding that harnesses the idea of Thomsen’s encoding and show that it is complete. In [2], Bundgaard et al. use the HOMER to translate the name-passing π -calculus. This translation is possible because a HOMER process can, in a way quite different from parameterization, operate names in the continuation processes (resources), and this allows flexibility so that names can be communicated in an intermediate fashion. In [11], Kouzapas et al. propose fully abstract encodings concerning first-order processes and session typed higher-order processes. Their encodings use session types to govern communications and show that in the context of session types, first-order and higher-order processes are equally expressive. This work is well related to those mentioned above (and that in this paper), though the context is quite different (i.e., session typed processes).

Despite the extensive research on encoding first-order processes with (variant) higher-order processes, the following question has remained open: *Is there an encoding of first-order processes by the higher-order processes with the capability of parameterization?* This question is important in two aspects. One is that parameterization brings about the core of lambda-calculus to higher-order concurrency, so it appears reasonable for such an extension to be able to express first-order processes which has long been shown to be capable of expressing the lambda-calculus. Knowing how this can be achieved would be interesting. The other is that the converse has a almost standard encoding method, i.e., encoding variants of higher-order processes with first-order processes. Yet higher-order processes are still short of an effective way to express first-order ones. Resolving this can also provide (technical) reference for practical work beyond the encoding itself.

Closely related with the first issue on expressiveness, the second issue this paper deals with is the characterization of bisimulation on higher-order processes. Bisimulation theory is a pivotal part of a process model, including the higher-order models, concerning which the almost standard behavioral equivalence is the context bisimulation [19]. The central idea of context bisimulation is that when comparing output actions, the transmitted process and the residual process (i.e., the process obtained after sending a process) are considered at the same time, rather than separately (like in the applicative higher-order bisimulation proposed by Thomsen [23,24]). For example (for simplicity we do not consider local

names), if P and Q are context bisimilar and $P \xrightarrow{\bar{a}A} P'$ (i.e., P outputs A on a and becomes P'), then $Q \xrightarrow{\bar{a}B} Q'$ (i.e., Q outputs B on a possibly involving some internal actions and becomes Q'), and for every (receiving) environment $E[\cdot]$, $P' | E[A]$ and $Q' | E[B]$ are still context bisimilar (here $|$ denotes concurrency, and $E[A]$ means putting A in the environment E). However, in its original form, context bisimulation suffers from inconvenience to use, because it calls for checking with regard to every possible receiving environment. This leads to works on the simpler characterization, called normal bisimulation, of the context bisimulation. The central idea of normal bisimulation, proposed by Sangiorgi [19, 22], is that instead of checking with a general process in input and a general context in output, one only needs to comply with the matching of some special process or context, specifically a class of terms called triggers. To meet this challenge, a crucial so-called factorization theorem is used to circumvent technical difficulty. We briefly explain how normal bisimulation is designed in the basic higher-order processes. In particular, the factorization states the following property, where \approx_{ct} denotes context bisimulation, and $\bar{m}.P$ and $m.P$ are CCS-like prefixes in which the communicated contents are not important [22].

$$E[A] \approx_{ct} (m)(E[\bar{m}.0] | !m.A)$$

One can clearly identify the reposition of the process A of interest, which in fact captures the core of the property: move A to a new position as a repository, which in turn can be retrieved as many times as needed in the original environment E , with the help of the pointer undertaken by the fresh channel m (called trigger). Inspired by the factorization, normal bisimulation can be developed. We take the output as an example (input is similar), and restriction operation in output is omitted for the sake of simplicity. As stated above, context bisimulation requires the following chasing diagram, which is now extended with an application of the factorization.

$$\begin{array}{ccccc} P & \cdots \approx_{ct} \cdots & Q \\ \bar{a}A \downarrow & & \downarrow \bar{a}B \\ P' & & Q' \end{array} \quad \begin{array}{c} P' | (m)(E[\bar{m}.0] | !m.A) \approx_{ct} P' | E[A] \\ \cdots \approx_{ct} \cdots \\ Q' | E[B] \approx_{ct} Q' | (m)(E[\bar{m}.0] | !m.B) \end{array}$$

Since context bisimulation \approx_{ct} is a congruence, one can cancel the common part of (the leftmost) $P' | (m)(E[\bar{m}.0] | !m.A)$ and (the rightmost) $Q' | (m)(E[\bar{m}.0] | !m.B)$, and simply requires that $P' | !m.A$ and $Q' | !m.B$ are related, without fearing losing any discriminating power. This in turn leads to the following requirement in normal bisimulation (assuming \mathcal{R} is a normal bisimulation).

$$\begin{array}{ccccc} P & \cdots \mathcal{R} \cdots & Q \\ \bar{a}A \downarrow & & \downarrow \bar{a}B \\ P' & & Q' \end{array} \quad \begin{array}{c} P' | !m.A \\ \cdots \mathcal{R} \cdots \\ Q' | !m.B \end{array}$$

Subsequent works attempt to extend the normal bisimulation to variants of higher-order processes. In Sangiorgi's initial work [19], normal bisimulation is also obtained for higher-order processes with parameterization. That characterization, however, is made in the presence of first-order processes (i.e., name-passing), and thus not very convincing with regard to the inner complexity of context bisimulation in presence of parameterization. In [26], we revisited this issue and show that in a purely higher-order setting (viz., no name-passing at all), parameterization on processes does not deprive one of the convenience of normal bisimulation. Although the idea is inspired by the original work of Sangiorgi, the proof approach is more direct. In [14, 15], Lenglet et al. study higher-order processes with passivation (i.e., the process in the output position may evolve), and report a normal bisimulation for a sub-calculus without the restriction operator, but that characterization has somewhat a different flavor, since the higher-order

bisimulation [24] rather than the context bisimulation is taken. Though these works carry out insightful research and give meaningful references, it is currently still not clear how to construct a simple characterization of context bisimulation based on parameterization over names, and this raises the following fundamental question: *Does higher-order processes with parameterization on names have a normal bisimulation?* In the second part of this paper, we move further from [19, 26], and offer a normal bisimulation for higher-order processes in the setting of parameterization over both names and processes.

Contribution In summary, our contribution of this work is as follows.

- We show that the extension with parameterization (on both names and processes) allows higher-order processes to interpret first-order processes in a surprisingly concise yet elegant manner. Such kind of encoding is of a somewhat dissimilar flavor, and moreover not possible in absence of parameterization. We give the detailed encoding strategy, and prove that it satisfies a number of desired properties well-known in the field.

The idea of the encoding in this paper is quite different from our abovementioned work in [28], where we build an encoding that allows parameterization merely on names (i.e., no parameterization on processes). The soundness of that encoding is not very satisfying, which in a sense defeats some purpose of the encoding, and this actually precipitates the work here.

- We establish the normal bisimulation, as an effectively simpler characterization of context bisimulation, for higher-order processes with both kinds of parameterization. This normal bisimulation extends those for higher-order processes without parameterization, particularly in the manipulation of abstractions on names. As far as we are concerned, similar characterization has not been reported before.

That the processes are purely higher-order (that is, without name-passing) improves the result in [19], and articulates that the characterization based on normal bisimulation is a property independent of first-order name-passing. Moreover, this does not contradict the argument in [26] that there is little hope that normal bisimulation exists in higher-order processes with (only) parameterization on names, because here the processes are capable of parameterization on processes as well (though still higher-order).

Organization The remainder of this paper is organized as below. In Section 2, we introduce the calculi and a notion of encoding used in this paper. In Section 3, we present the encoding from first-order processes to higher-order processes with parameterization, and discuss its properties. In Section 4, we define the normal bisimulation for higher-order processes with parameterization, and prove that it truly characterizes context bisimulation. Section 5 concludes this work and point out some further directions.

2 Preliminary

In this section we present the basic definitions and notations used in this work.

2.1 Calculus π

The first-order (name-passing) pi-calculus, π , is proposed by Milner et al. [17]. For the sake of simplicity, throughout the paper, names (ranged over by m, n, u, v, w) are divided into two classes: name constants (ranged over by a, b, c, d, e) and name variables (ranged over by x, y, z) [3, 4, 6]. The grammar is as below with the constructs having their standard meaning. We note that guarded input replication is used instead of general replication, and this does not decrease the expressiveness [21] [7].

$$P, Q := 0 \mid m(x).P \mid \overline{m}n.P \mid (c)P \mid P \mid Q \mid !m(x).P$$

A name constant a is bound (or local) in $(a)P$ and free (or global) otherwise. A name variable x is bound in $a(x).P$ and free otherwise. Respectively $\text{fn}(\cdot), \text{bn}(\cdot), \text{n}(\cdot), \text{fnv}(\cdot), \text{bnv}(\cdot), \text{nv}(\cdot)$ denote free name constants, bound name constants, names, free name variables, bound name variables, and name variables in a set of processes. A name is fresh if it does not appear in any process under discussion. By default, closed processes are considered, i.e., those having no free variables. As usual, here are a few derived operators: $\bar{a}(d).P \stackrel{\text{def}}{=} (d)\bar{a}d.P$, $a.P \stackrel{\text{def}}{=} a(x).P$ ($x \notin \text{fv}(P)$), $\bar{a}.P \stackrel{\text{def}}{=} \bar{a}(d).P$ ($d \notin \text{fn}(P)$); $\tau.P \stackrel{\text{def}}{=} (a)(a.P \mid \bar{a}.0)$ (a fresh). A trailing 0 process is usually omitted. We denote tuples by a tilde. For tuple \tilde{n} : $|\tilde{n}|$ denotes its length; $m\tilde{n}$ denotes incorporating m . Multiple restriction $(c_1)(c_2)\cdots(c_k)E$ is abbreviated as $(\tilde{c})E$. Substitution $P\{n/m\}$ is a mapping that replaces m with n in P while keeping the rest unchanged. A context C is a process with some subprocess replaced by the hole $[\cdot]$, and $C[A]$ is the process obtained by filling in the hole by A .

The semantics of π is defined by the LTS (Labelled Transition System) below.

$$\begin{array}{c} \frac{}{a(x).P \xrightarrow{a(b)} P\{b/x\}} \quad \frac{}{\bar{a}b.P \xrightarrow{\bar{a}b} P} \quad \frac{}{!a(x).P \xrightarrow{a(b)} P\{b/x\} \mid !a(x).P} \quad \frac{P \xrightarrow{\lambda} P' \quad c \notin n(\lambda)}{(c)P \xrightarrow{\lambda} P'} \\[10pt] \frac{P \xrightarrow{\bar{a}c} P' \quad c \neq a}{(c)P \xrightarrow{\bar{a}(c)} P'} \quad \frac{P \xrightarrow{\lambda} P' \quad \text{bn}(\lambda) \cap \text{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\lambda} P' \mid Q} \quad \frac{P \xrightarrow{a(b)} P' \quad Q \xrightarrow{\bar{a}b} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \quad \frac{P \xrightarrow{a(b)} P' \quad Q \xrightarrow{\bar{a}(b)} Q'}{P \mid Q \xrightarrow{\tau} (b)(P' \mid Q')} \end{array}$$

Actions, ranged over by λ, α , comprise internal move τ , and visible ones: input $(a(b))$, output $(\bar{a}b)$ and bound output $(\bar{a}(c))$. We note that actions occur only on name constants, and a communicated name is also a constant. We denote by \equiv the standard structural congruence [17] [22], which is the smallest relation satisfying the monoid laws for parallel composition, commutative laws for both composition and restriction, and a distributive law $(c)(P \mid Q) \equiv (c)P \mid Q$ (if $c \notin \text{fn}(Q)$). We use \Longrightarrow for the reflexive transitive closure of $\xrightarrow{\tau}, \xrightarrow{\lambda}$ for $\Longrightarrow \xrightarrow{\lambda} \Longrightarrow$, and $\xRightarrow{\hat{\lambda}}$ for $\xrightarrow{\lambda}$ if λ is not τ , and \Longrightarrow otherwise. A process P is divergent, denoted P^\dagger , if it has an infinite sequence of τ actions.

Throughout the paper, we use the following standard notion of ground bisimulation [4, 17, 22].

Definition 1. A ground bisimulation is a symmetric relation \mathcal{R} on π processes s.t. whenever $P \mathcal{R} Q$ the following property holds: If $P \xrightarrow{\alpha} P'$ where α is $a(b)$, $\bar{a}b$, $\bar{a}(b)$, or τ , then $Q \xRightarrow{\hat{\alpha}} Q'$ for some Q' and $P' \mathcal{R} Q'$. Ground bisimilarity, \approx_g , is the largest ground bisimulation.

We denote by \sim_g the strong ground bisimilarity (i.e., replacing $\xRightarrow{\hat{\alpha}}$ with $\xrightarrow{\alpha}$ in the definition). It is well-known that \approx_g is a congruence [4, 22], and coincides with the so-called local bisimilarity as defined below [5, 25].

Definition 2. Local bisimilarity \approx_l is the largest symmetric local bisimulation relation \mathcal{R} on π processes such that: (1) if $P \xrightarrow{\lambda} P'$, λ is not bound output, then $Q \xRightarrow{\hat{\lambda}} Q'$ and $P' \mathcal{R} Q'$; (2) if $P \xrightarrow{\bar{a}(b)} P'$, then $Q \xRightarrow{\bar{a}(b)} Q'$, and for every R , $(b)(P' \mid R) \mathcal{R} (b)(Q' \mid R)$.

2.2 Calculus $\Pi^{D,d}$

For the sake of conciseness, we first define the basic higher-order calculus and then the extension with parameterizations.

2.2.1 Calculus Π

The basic higher-order (process-passing) calculus, Π , is defined by the following grammar in which the operators have their standard meaning. We denote by X, Y, Z process variables.

$$T, T' ::= 0 \mid X \mid u(X).T \mid \bar{u}T'.T \mid T \mid T' \mid (c)T \mid !u(X).T \mid !\bar{u}T'.T$$

We use $a.0$ for $a(X).0$, $\bar{a}.0$ for $\bar{a}0.0$, $\tau.P$ for $(a)(a.P|\bar{a}.0)$, and sometimes $\bar{a}[A].T$ for $\bar{a}A.T$. Like π , a tilde represents a tuple. We reuse the notations for names in π and additionally use $\text{fpv}(\cdot)$, $\text{bpv}(\cdot)$, $\text{pv}(\cdot)$ respectively to denote free process variables, bound process variables and process variables in a set of processes. Closed processes are those having no free variables. A higher-order substitution $T\{A/X\}$ replaces variable X with A and can be extended to tuples in the usual way. $E[\tilde{X}]$ denotes E with (possibly) variables \tilde{X} , and $E[\tilde{A}]$ stands for $E\{\tilde{A}/\tilde{X}\}$. The guarded replications used in the grammar can actually be derived [13, 24], and we make them primitive for convenience. The semantics of Π is as below.

$$\begin{array}{c} \frac{}{a(X).T \xrightarrow{a(A)} T\{A/X\}} \quad \frac{}{\bar{a}A.T \xrightarrow{\bar{a}A} T} \quad \frac{T \xrightarrow{\lambda} T'}{(c)T \xrightarrow{\lambda} (c)T'}^{c \notin n(\lambda)} \quad \frac{T \xrightarrow{\lambda} T'}{T | T_1 \xrightarrow{\lambda} T' | T_1} \quad \frac{}{!aA.T \xrightarrow{\bar{a}A} T | !aA.T} \\[10pt] \frac{T \xrightarrow{(\tilde{c})\bar{a}[A]} T'}{(d)T \xrightarrow{(\tilde{c})\bar{a}[A]} T'}^{d \in fn(A) - \{\tilde{c}, a\}} \quad \frac{T_1 \xrightarrow{a(A)} T'_1, T_2 \xrightarrow{(\tilde{c})\bar{a}[A]} T'_2}{T_1 | T_2 \xrightarrow{\tau} (\tilde{c})(T'_1 | T'_2)} \quad \frac{}{!a(X).T \xrightarrow{a(A)} T\{A/X\} | !a(X).T} \end{array}$$

We denote by α, λ the actions: internal move (τ), input ($a(A)$), output ($(\tilde{c})\bar{a}A$) in which \tilde{c} is some local names carried by A during the output. We always assume no name capture with resort to α -conversion.

The notations $\Rightarrow, \xRightarrow{\lambda}$ and $\xRightarrow{\hat{\lambda}}$ are similar to those in π . We also reuse \equiv for the structural congruence in Π (and also $\Pi^{D,d}$ to be defined shortly) [22], and this shall not raise confusion under specific context.

2.2.2 Calculus $\Pi^{D,d}$

Parameterization extends Π with the syntax and semantics below. Symbol U_i (respectively, K_i) ($i = 1, \dots, n$) is used as a meta-parameter of an abstraction (respectively, meta-instance of an application), and stands for a process variable or name variable (respectively, a process or a name).

$$\begin{array}{ll} \text{Extension of syntax:} & \langle U_1, U_2, \dots, U_n \rangle T \mid T' \langle K_1, K_2, \dots, K_n \rangle \\ \text{Extension of semantics:} & \frac{Q \equiv P \quad P \xrightarrow{\lambda} P' \quad P' \equiv Q'}{Q \xrightarrow{\lambda} Q'} \\ \text{Extension of structural congruence } (\equiv): & F \langle \tilde{K} \rangle \equiv T \{ \tilde{K} / \tilde{U} \} \quad \text{where } F \stackrel{\text{def}}{=} \langle \tilde{U} \rangle T \text{ and } |\tilde{U}| = |\tilde{K}| \end{array}$$

We denote by $\langle U_1, U_2, \dots, U_n \rangle T$ an n -ary abstraction in which U_1, U_2, \dots, U_n are the parameters to be instantiated during the application $T' \langle K_1, K_2, \dots, K_n \rangle$ in which the parameters are replaced by instances K_1, K_2, \dots, K_n . This application is modelled by an extensional rule for structural congruence as above, in combination with the usual LTS rule for structural congruence as well, so as to make the process engaged in application evolve effectively. The condition $|\tilde{U}| = |\tilde{K}|$ requires that the parameters and the instantiating objects should be equal in length.

Now parameterization on process is obtained by taking \tilde{U}, \tilde{K} as \tilde{X}, \tilde{T}' respectively, and parameterization on names is obtained by taking \tilde{U}, \tilde{K} as \tilde{x}, \tilde{u} respectively. The corresponding abstractions are sometimes called process abstraction and name abstraction respectively. For convenience, names are handled in the same way as that in π (so are the related notations). We denote by $\Pi^{D,d}$ the calculus Π extended with both kinds of parameterizations. Calculus $\Pi^{D,d}$ can be made more precise with the help of a type system [19] which however is not important for this work and not presented. We note that in $\langle U_1, U_2, \dots, U_n \rangle T$, variables U_1, U_2, \dots, U_n are bound.

Throughout the paper, we rely on the following notion of context bisimulation [19, 20].

Definition 3. A symmetric relation \mathcal{R} on $\Pi^{D,d}$ processes is a context bisimulation, if $P \mathcal{R} Q$ implies the following properties: (1) if $P \xrightarrow{\alpha} P'$ and α is $a(A)$ or τ , then $Q \xRightarrow{\hat{\alpha}} Q'$ for some Q' and $P' \mathcal{R} Q'$;

(2) if $P \xrightarrow{(\tilde{c})\bar{a}A} P'$ and A is a process abstraction or name abstraction or not an abstraction, then $Q \xrightarrow{(\tilde{d})\bar{a}B} Q'$

for some B that is accordingly a process abstraction or name abstraction or not an abstraction, and moreover for every $E[X]$ s.t. $\{\tilde{c}, \tilde{d}\} \cap fn(E) = \emptyset$ it holds that $(\tilde{c})(E[A] \mid P') \not\mathcal{R} (\tilde{d})(E[B] \mid Q')$. Context bisimilarity, written \approx_{ct} , is the largest context bisimulation.

We note that the matching for output in context bisimulation is required to bear the same kind of communicated process as compared to the simulated action. Relation \sim_{ct} denotes the strong context bisimilarity. As is well-known, \approx_{ct} is a congruence [19, 20].

2.3 A notion of encoding

We define a notion of encoding in this section. We assume a process model \mathcal{L} is a triplet $(\mathcal{P}, \rightarrow, \approx)$, where \mathcal{P} is the set of processes, \rightarrow is the LTS with a set \mathcal{A} of actions, and \approx is a behavioral equivalence. Given $\mathcal{L}_i \stackrel{\text{def}}{=} (\mathcal{P}_i, \rightarrow_i, \approx_i)$ ($i=1,2$), an encoding from \mathcal{L}_1 to \mathcal{L}_2 is a function $\llbracket \cdot \rrbracket : \mathcal{P}_1 \rightarrow \mathcal{P}_2$ that satisfies some set of criteria. Notation $\llbracket \mathcal{P}_1 \rrbracket$ stands for the image of the \mathcal{L}_1 -processes inside \mathcal{L}_2 under the encoding. It should be clear that $\llbracket \mathcal{P}_1 \rrbracket \subseteq \mathcal{P}_2$. We use \approx_2 to denote the behavioural equivalence \approx_2 restricted to $\llbracket \mathcal{P}_1 \rrbracket$ [10]. The following criteria set (Definition 4) used in this paper stems from [12] (the variant [12] provides is based on [8]). As is known, encodability enjoys transitivity [12]. We will show that the encoding in Section 3 satisfies all the criteria in Definition 4 except adequacy (1a).

Definition 4 (Criteria for encodings). **Static criteria:** (1) Compositionality. *For any k -ary operator op of \mathcal{L}_1 , and all $P_1, \dots, P_k \in \mathcal{P}_1$, $\llbracket op(P_1, \dots, P_k) \rrbracket = C_{op}[\llbracket P_1 \rrbracket, \dots, \llbracket P_k \rrbracket]$ for some (multihole) context $C_{op}[\dots] \in \mathcal{P}_2$;*

Dynamic criteria: (1a) Adequacy. *$P \approx_1 P'$ implies $\llbracket P \rrbracket \approx_2 \llbracket P' \rrbracket$. This is also known as soundness. The converse is known as completeness;* (1b) Weak adequacy (or weak soundness). *$P \approx_1 P'$ implies $\llbracket P \rrbracket \approx_2 \llbracket P' \rrbracket$;* (2) Divergence-reflecting. *If $\llbracket P \rrbracket$ diverges, so does P .*

Adequacy (1a) obviously entails weak adequacy (1b), since \approx_2 allows more processes in the target model \mathcal{L}_2 (thus more variety of contexts). Yet weak adequacy is still useful because it may be too strong if one requires the encoding process to be compatible with all kinds of contexts in the target model. For instance, in order to achieve first-order interactions in a higher-order target model, it appears quite demanding to require equivalence under all kinds of input because the target higher-order model may have more powerful computation ability (so it can feed a much involved input). So sometimes using limited contexts in the target model may be sufficient to meet the goal of the encoding.

It is worthwhile to note that the criteria are short of those for operational correspondence. Although generally soundness and completeness appear not very informative in absence of operational correspondence (and the others) [9, 18], arguably we make this choice in this work out of the following consideration. The criteria for operational correspondence used in [8], though proven useful in many models, appear not quite convenient when discussing encodings into higher-order models [12], since (for example) the case of input can be hard to comply with the criteria due to the increased complexity in the environment (namely in the context of the target higher-order model). After all, here it seems more important to have the soundness and completeness properties eventually (w.r.t. the canonical bisimulation equivalences in the source and target models), likely in a different manner of operational correspondence. Notwithstanding, we will discuss the operational correspondence of the encoding in Section 3. Moreover, as will be seen, the concrete operational correspondence in there somehow strengthens the criteria of operational correspondence (and related concepts) used in [8, 12] (in [8] the criteria are not action-labelled and thus the notion of success sensitiveness is contrived; in [12] a labelled variant criteria is posited to its purpose). Beyond the scope of this paper, it would be intriguing to examine the possibility of formally pinning down some variant criteria of operational correspondence having vantage for higher-order (process) models.

3 Encoding π into $\Pi^{D,d}$

We show that π can be encoded in $\Pi^{D,d}$.

3.1 The encoding

We have the encoding defined as below (being homomorphism on the other operators, except that the encoding of input guarded replication is defined as $\llbracket !m(x).P \rrbracket \stackrel{\text{def}}{=} !\llbracket m(x).P \rrbracket$).

$$\begin{aligned} \llbracket m(x).P \rrbracket &\stackrel{\text{def}}{=} m(Y).Y \langle \langle x \rangle \rrbracket \llbracket P \rrbracket \rangle \\ \llbracket \bar{m}n.Q \rrbracket &\stackrel{\text{def}}{=} \bar{m}[\langle Z \rangle \langle Z \langle n \rangle \rangle].\llbracket Q \rrbracket \end{aligned}$$

The encoding above uses both name parameterization and process parameterization. Typically one can assume that Y and Z are fresh for simplicity, but this is not essential, because these variables are bound and can be α -converted whenever necessary, and moreover the encoded first-order process does not have higher-order variables. Specifically, the encoding of an output ‘transmits’ the name to be sent (i.e., n) in terms of a process parameterization (i.e., $\langle Z \rangle \langle Z \langle n \rangle \rangle$) that, once being received by the encoding of an input, is instantiated by a name-parameterized term (i.e., $\langle x \rangle \llbracket P \rrbracket$), which then can apply n on x in the encoding of P , thus fulfilling ‘name-passing’. Below we give an example. Suppose $P \stackrel{\text{def}}{=} (c)(a(x).\bar{x}c.P_1)$ and $Q \stackrel{\text{def}}{=} (d)(\bar{a}d.d(y).Q_1)$. So

$$\begin{aligned} P | Q &\xrightarrow{\tau} (d)((c)(\bar{d}c.P_1\{d/x\}) | d(y).Q_1) \\ &\xrightarrow{\tau} (dc)(P_1\{d/x\} | Q_1\{c/y\}) \end{aligned}$$

The encoding and interactions of $\llbracket P | Q \rrbracket$ are as below. For clarity, we use **bold font** to indicate the evolving part during a communication.

$$\begin{aligned} \llbracket P | Q \rrbracket &\equiv (c)(a(Y).Y \langle \langle x \rangle \rrbracket \llbracket \bar{x}c.P_1 \rrbracket \rangle) | (d)(\bar{a}[\langle Z \rangle \langle Z \langle d \rangle \rangle].\llbracket d(y).Q_1 \rrbracket) \\ &\xrightarrow{\tau} (d)((c)((\langle \mathbf{Z} \rangle \langle \mathbf{Z} \langle \mathbf{d} \rangle \rangle) \langle \langle \mathbf{x} \rangle \rrbracket \llbracket \bar{x}c.P_1 \rrbracket \rangle) | \llbracket d(y).Q_1 \rrbracket) \\ &\equiv (d)((c)(\llbracket \bar{x}c.P_1 \rrbracket \{ \mathbf{d}/\mathbf{x} \}) | \llbracket d(y).Q_1 \rrbracket) \\ &\equiv (d)((c)((\bar{x}[\langle \mathbf{Z} \rangle \langle \mathbf{Z} \langle \mathbf{c} \rangle \rangle].\llbracket P_1 \rrbracket) \{ \mathbf{d}/\mathbf{x} \}) | d(Y).Y \langle \langle \mathbf{y} \rangle \rrbracket \llbracket Q_1 \rrbracket \rangle) \\ &\equiv (d)((c)((\bar{d}[\langle \mathbf{Z} \rangle \langle \mathbf{Z} \langle \mathbf{c} \rangle \rangle].\llbracket P_1 \rrbracket \{ \mathbf{d}/\mathbf{x} \}) | d(Y).Y \langle \langle \mathbf{y} \rangle \rrbracket \llbracket Q_1 \rrbracket \rangle) \\ &\xrightarrow{\tau} (dc)(\llbracket P_1 \rrbracket \{ \mathbf{d}/\mathbf{x} \} | (\langle \mathbf{Z} \rangle \langle \mathbf{Z} \langle \mathbf{c} \rangle \rangle) \langle \langle \mathbf{y} \rangle \rrbracket \llbracket Q_1 \rrbracket \rangle) \\ &\equiv (dc)(\llbracket P_1 \rrbracket \{ \mathbf{d}/\mathbf{x} \} | \llbracket Q_1 \rrbracket \{ \mathbf{c}/\mathbf{y} \}) \\ &\equiv (dc)(\llbracket P_1 \rrbracket \{ \mathbf{d}/\mathbf{x} \} | \llbracket Q_1 \rrbracket \{ \mathbf{c}/\mathbf{y} \}) \end{aligned}$$

Apparently the encoding is compositional, preserves the (free) names, and moreover divergence-reflecting (since the encoding does not introduce any extra internal action), as stated in the follow-up lemma whose proof is a standard induction.

Lemma 5. Assume P is a π process. The encoding above from π to $\Pi^{D,d}$ is compositional and divergence-reflecting; moreover $\llbracket P \rrbracket \{n/m\} \equiv \llbracket P \rrbracket \{n/m\}$.

3.2 Operational correspondence

We have the following properties clarifying the correspondence of actions before and after the encoding. To delineate some case of the operational correspondence in terms of certain special input, i.e., a

trigger, we define $Tr_m^D \stackrel{\text{def}}{=} \langle Z \rangle \bar{m}Z$ in which m is assumed to be fresh (it will also be used in Section 4, but here simply allows for more flexible characterization of the operational correspondence). We note that sometimes existential quantification is omitted when it is clear from context.

Lemma 6. *Suppose P is a π process. (1) If $P \xrightarrow{a(b)} P'$, then $\llbracket P \rrbracket \xrightarrow{a(\langle Z \rangle \langle Z(b) \rangle)} T$ and $T \sim_{ct} \llbracket P' \rrbracket$; (2) If $P \xrightarrow{a(b)} P'$, then $\llbracket P \rrbracket \xrightarrow{a(Tr_m^D)} T$ and $(m)(T \mid !m(Y).Y \langle b \rangle) \approx_{ct} \llbracket P' \rrbracket$; (3) If $P \xrightarrow{\bar{a}b} P'$, then $\llbracket P \rrbracket \xrightarrow{\bar{a}[\langle Z \rangle \langle Z(b) \rangle]} T$ and $T \sim_{ct} \llbracket P' \rrbracket$; (4) If $P \xrightarrow{\bar{a}(b)} P'$, then $\llbracket P \rrbracket \xrightarrow{(b)\bar{a}[\langle Z \rangle \langle Z(b) \rangle]} T$ and $T \sim_{ct} \llbracket P' \rrbracket$; (5) If $P \xrightarrow{\tau} P'$, then $\llbracket P \rrbracket \xrightarrow{\tau} T$ and $T \sim_{ct} \llbracket P' \rrbracket$.*

The converse is as below.

Lemma 7. *Suppose P is a π process. (1) If $\llbracket P \rrbracket \xrightarrow{a(\langle Z \rangle \langle Z(b) \rangle)} T$, then $P \xrightarrow{a(b)} P'$ and $T \sim_{ct} \llbracket P' \rrbracket$; (2) If $\llbracket P \rrbracket \xrightarrow{a(Tr_m^D)} T$, then $P \xrightarrow{a(b)} P'$ and $(m)(T \mid !m(Y).Y \langle b \rangle) \approx_{ct} \llbracket P' \rrbracket$; (3) If $\llbracket P \rrbracket \xrightarrow{\bar{a}[\langle Z \rangle \langle Z(b) \rangle]} T$, then $P \xrightarrow{\bar{a}b} P'$ and $T \sim_{ct} \llbracket P' \rrbracket$; (4) If $\llbracket P \rrbracket \xrightarrow{(b)\bar{a}[\langle Z \rangle \langle Z(b) \rangle]} T$, then $P \xrightarrow{\bar{a}(b)} P'$ and $T \sim_{ct} \llbracket P' \rrbracket$; (5) If $\llbracket P \rrbracket \xrightarrow{\tau} T$, then $P \xrightarrow{\tau} P'$ and $T \sim_{ct} \llbracket P' \rrbracket$.*

Lemma 6 and Lemma 7 can be proven in a similar fashion (details can be found in [27]), and moreover be lifted to the weak situation. That is, if one replaces strong transitions (single arrows) with weak transitions (double arrows), the results still hold (\sim_{ct} retains because the encoding does not bring any extra internal action); see [19, 22] for a reference. We will however simply refer to these two lemmas in related discussions.

3.3 Soundness

In this section, we discuss the soundness of the encoding. First of all, it is unfortunate that the soundness of the encoding is not true. To see this, take the processes R_1 and R_2 below. We recall that the CCS-like prefixes are defined as usual, i.e., $a.P \stackrel{\text{def}}{=} a(x).P$ ($x \notin n(P)$), $\bar{a}.P \stackrel{\text{def}}{=} (c)\bar{a}c.P$ ($c \notin n(P)$); sometimes we trim the trailing 0, e.g., a stands for $a.0$ and \bar{a} for $\bar{a}.0$.

$$R_1 \stackrel{\text{def}}{=} (b)(a.\bar{b} \mid b.\bar{c}) \quad R_2 \stackrel{\text{def}}{=} (b)(a.\bar{b} \mid b.\bar{c} \mid b.\bar{c})$$

Obviously, R_1 and R_2 are ground bisimilar. Now we examine their encodings.

$$\begin{aligned} \llbracket R_1 \rrbracket &\equiv (b)(a(Y).Y \langle \langle x \rangle \llbracket \bar{b} \rrbracket \rangle \mid b(Y).Y \langle \langle x \rangle \llbracket \bar{c} \rrbracket \rangle) \\ \llbracket R_2 \rrbracket &\equiv (b)(a(Y).Y \langle \langle x \rangle \llbracket \bar{b} \rrbracket \rangle \mid b(Y).Y \langle \langle x \rangle \llbracket \bar{c} \rrbracket \rangle \mid b(Y).Y \langle \langle x \rangle \llbracket \bar{c} \rrbracket \rangle) \end{aligned}$$

We show that $\llbracket R_1 \rrbracket$ and $\llbracket R_2 \rrbracket$ are not context bisimilar. Define $T \stackrel{\text{def}}{=} (m)(\bar{a}[\langle Z \rangle \bar{m}Z] \mid m(X).(X \langle d \rangle \mid X \langle d \rangle)$. Then $(a)(\llbracket R_1 \rrbracket \mid T)$ and $(a)(\llbracket R_2 \rrbracket \mid T)$ can be distinguished. The latter can fire two output on c , whereas the former cannot, as shown below.

$$\begin{aligned} (a)(\llbracket R_1 \rrbracket \mid T) &\xrightarrow{\tau} \sim_{ct} (m)((b)(\bar{m}[\langle x \rangle \llbracket \bar{b} \rrbracket] \mid b(Y).Y \langle \langle x \rangle \llbracket \bar{c} \rrbracket \rangle) \mid m(X).(X \langle d \rangle \mid X \langle d \rangle)) \\ &\xrightarrow{\tau} \sim_{ct} (b)(b(Y).Y \langle \langle x \rangle \llbracket \bar{c} \rrbracket \rangle \mid \llbracket \bar{b} \rrbracket \mid \llbracket \bar{b} \rrbracket) \\ &\equiv (b)(b(Y).Y \langle \langle x \rangle \llbracket \bar{c} \rrbracket \rangle \mid (e)\bar{b}[\langle Z \rangle \langle Z(e) \rangle] \mid \llbracket \bar{b} \rrbracket) \\ &\xrightarrow{\tau} \sim_{ct} (b)(\llbracket \bar{c} \rrbracket \mid \llbracket \bar{b} \rrbracket) \\ &\equiv (b)((f)\bar{c}[\langle Z \rangle \langle Z(f) \rangle] \mid \llbracket \bar{b} \rrbracket) \\ &\xrightarrow{(f)\bar{c}[\langle Z \rangle \langle Z(f) \rangle]} \sim_{ct} 0 \end{aligned}$$

$$\begin{aligned}
(a)(\llbracket R_2 \rrbracket \mid T) &\xrightarrow{\tau} \sim_{ct} (m)((b)(\overline{m}[\langle x \rangle \llbracket \overline{b} \rrbracket] \mid b(Y).Y \langle \langle x \rangle \llbracket \overline{c} \rrbracket \rangle \mid b(Y).Y \langle \langle x \rangle \llbracket \overline{c} \rrbracket \rangle) \mid m(X).(X \langle d \rangle \mid X \langle d \rangle)) \\
&\xrightarrow{\tau} \sim_{ct} (b)(b(Y).Y \langle \langle x \rangle \llbracket \overline{c} \rrbracket \rangle \mid b(Y).Y \langle \langle x \rangle \llbracket \overline{c} \rrbracket \rangle \mid \llbracket \overline{b} \rrbracket \mid \llbracket \overline{b} \rrbracket) \\
&\equiv (b)(b(Y).Y \langle \langle x \rangle \llbracket \overline{c} \rrbracket \rangle \mid b(Y).Y \langle \langle x \rangle \llbracket \overline{c} \rrbracket \rangle \mid (e)\overline{b}[\langle Z \rangle(Z \langle e \rangle)] \mid (e)\overline{b}[\langle Z \rangle(Z \langle e \rangle)]) \\
&\xrightarrow{\tau} \sim_{ct} \llbracket \overline{c} \rrbracket \mid \llbracket \overline{c} \rrbracket \\
&\equiv (f)\overline{c}[\langle Z \rangle(Z \langle f \rangle)] \mid (f)\overline{c}[\langle Z \rangle(Z \langle f \rangle)] \\
&\xrightarrow{(f)\overline{c}[\langle Z \rangle(Z \langle f \rangle)]} \sim_{ct} (f)\overline{c}[\langle Z \rangle(Z \langle f \rangle)] \\
&\xrightarrow{(f)\overline{c}[\langle Z \rangle(Z \langle f \rangle)]} \sim_{ct} 0
\end{aligned}$$

Intuitively, the reason general soundness does not hold is that context bisimulation is somewhat more discriminating in the target higher-order calculus, which can have more flexibility when dealing with blocks of processes in presence of parameterization (e.g., some subprocess can be sent as needed). This is however beyond the capability of a first-order process.

In spite of the falsity of soundness in general, we can have a somewhat weaker yet still sensible soundness. Remember that our main goal is to achieve first-order concurrency in the higher-order model, so maybe we do not need to be so demanding when coping with the encodings of first-order processes, that is, when testing an encoding process with an input, one can focus on those representing a name instead of a general one. Then it is expected that soundness will hold under this assumption. Fortunately, this is indeed true.

We have the following lemma stating the weak soundness of the encoding. Recall that \approx_{ct} is the \sim_{ct} restricted to the image of the encoding (i.e., the processes in the target model that have reverse-image w.r.t. the encoding).

Lemma 8. *Suppose P is a π process. Then $P \approx_g Q$ implies $\llbracket P \rrbracket \approx_{ct} \llbracket Q \rrbracket$.*

Proof. We show that $\mathcal{R} \stackrel{\text{def}}{=} \{(\llbracket P \rrbracket, \llbracket Q \rrbracket) \mid P \approx_g Q\} \cup \approx_{ct}$ is a context bisimulation up-to context and \sim_{ct} (we refer the reader to, for example, [1, 22] and the references therein for the up-to proof technique for establishing bisimulations; we note that using \sim_{ct} here is sufficient since it is stronger than \approx_{ct} , i.e., \sim_{ct} restricted to the image of the encoding).

Suppose $\llbracket P \rrbracket \mathcal{R} \llbracket Q \rrbracket$. There are several cases, where Lemma 6 and Lemma 7 play an important part.

- $\llbracket P \rrbracket \xrightarrow{a(\langle Z \rangle(Z \langle b \rangle))} T$. By Lemma 7, $P \xrightarrow{a(b)} P'$ and $T \sim_{ct} \llbracket P' \rrbracket$. Because $P \approx_g Q$, we know that $Q \xrightarrow{a(b)} Q' \approx_g P'$ and thus $\llbracket P' \rrbracket \mathcal{R} \llbracket Q' \rrbracket$. Then by Lemma 6, $\llbracket Q \rrbracket \xrightarrow{a(\langle Z \rangle(Z \langle b \rangle))} T'$ and $T' \sim_{ct} \llbracket Q' \rrbracket$. So we have $T \sim_{ct} \llbracket P' \rrbracket \mathcal{R} \llbracket Q' \rrbracket \sim_{ct} T'$.
- $\llbracket P \rrbracket \xrightarrow{\overline{a}(b)\langle Z \rangle(Z \langle b \rangle)} T$. By Lemma 7, $P \xrightarrow{\overline{a}(b)} P'$ and $T \sim_{ct} \llbracket P' \rrbracket$. Because $P \approx_g Q$, we know that $Q \xrightarrow{\overline{a}(b)} Q' \approx_g P'$ and thus $\llbracket P' \rrbracket \mathcal{R} \llbracket Q' \rrbracket$. Then by Lemma 6, $\llbracket Q \rrbracket \xrightarrow{(b)\overline{a}[\langle Z \rangle(Z \langle b \rangle)]} T'$ and $T' \sim_{ct} \llbracket Q' \rrbracket$. Consider the following pair

$$(b)(T \mid E[A]) \quad , \quad (b)(T' \mid E[A])$$

in which $b \notin \text{fn}(E[X])$ and $A \stackrel{\text{def}}{=} \langle Z \rangle(Z \langle b \rangle)$. So

$$(b)(T \mid E[A]) \sim_{ct} (b)(\llbracket P' \rrbracket \mid E[A]) \quad , \quad (b)(\llbracket Q' \rrbracket \mid E[A]) \sim_{ct} (b)(T' \mid E[A])$$

By setting a context $C \stackrel{\text{def}}{=} (b)([\cdot] \mid E[A])$, we have the following pair in which $\llbracket P' \rrbracket \mathcal{R} \llbracket Q' \rrbracket$.

$$C[\llbracket P' \rrbracket] \quad , \quad C[\llbracket Q' \rrbracket]$$

This suffices to close this case in terms of the up-to context requirement.

- $\llbracket P \rrbracket \xrightarrow{\bar{a}[\langle Z \rangle(Z\langle b \rangle)]} T$. This case is similar to the last case.
- $\llbracket P \rrbracket \xrightarrow{\tau} T$. By Lemma 7, $P \xrightarrow{\tau} P'$ and $T \sim_{ct} \llbracket P' \rrbracket$. From $P \approx_g Q$, we know $Q \Rightarrow Q' \approx_g P'$ and thus $\llbracket P' \rrbracket \mathcal{R} \llbracket Q' \rrbracket$. Then by Lemma 6, $\llbracket Q \rrbracket \Rightarrow T'$ and $T' \sim_{ct} \llbracket Q' \rrbracket$. So we have $T \sim_{ct} \llbracket P' \rrbracket \mathcal{R} \llbracket Q' \rrbracket \sim_{ct} T'$. \square

3.4 Completeness

The completeness of the encoding is stated in the lemma below. We note that completeness is true even if we do not constrain the domain to be the image of the encoded π processes.

Lemma 9. *Suppose P is a π process. Then $\llbracket P \rrbracket \approx_{ct} \llbracket Q \rrbracket$ implies $P \approx_g Q$.*

Proof. We show that $\mathcal{R} \stackrel{\text{def}}{=} \{(P, Q) \mid \llbracket P \rrbracket \approx_{ct} \llbracket Q \rrbracket\} \cup \approx_g$ is a local bisimulation. Suppose $P \mathcal{R} Q$. There are several cases.

- $P \xrightarrow{a(b)} P'$. By Lemma 6, $\llbracket P \rrbracket \xrightarrow{a[\langle Z \rangle(Z\langle b \rangle)]} T$ and $T \sim_{ct} \llbracket P' \rrbracket$. Because $\llbracket P \rrbracket \approx_{ct} \llbracket Q \rrbracket$, we know that $\llbracket Q \rrbracket \xrightarrow{a[\langle Z \rangle(Z\langle b \rangle)]} T' \sim_{ct} T$. By Lemma 7, $Q \xrightarrow{a(b)} Q'$ and $T' \sim_{ct} \llbracket Q' \rrbracket$. Thus we have $\llbracket P' \rrbracket \sim_{ct} T \approx_{ct} T' \sim_{ct} \llbracket Q' \rrbracket$, so $P' \mathcal{R} Q'$, which fulfills this case.
- $P \xrightarrow{\bar{a}b} P'$. By Lemma 6, $\llbracket P \rrbracket \xrightarrow{\bar{a}[\langle Z \rangle(Z\langle b \rangle)]} T$ and $T \sim_{ct} \llbracket P' \rrbracket$. Since $\llbracket P \rrbracket \approx_{ct} \llbracket Q \rrbracket$, we know that $\llbracket P \rrbracket$ must be able to be matched by $\llbracket Q \rrbracket \xrightarrow{\bar{a}[\langle Z \rangle(Z\langle b \rangle)]} T'$, because $\llbracket Q \rrbracket$ can only output such shape of processes, and if the matching is, e.g., $\llbracket Q \rrbracket \xrightarrow{\bar{a}[\langle Z \rangle(Z\langle c \rangle)]} T''$ then a context can be designed to distinguish between $\llbracket P \rrbracket$ and $\llbracket Q \rrbracket$. So for every $E[X]$, we have $T \mid E[\langle Z \rangle(Z\langle b \rangle)] \approx_{ct} T' \mid E[\langle Z \rangle(Z\langle b \rangle)]$. By Lemma 7, $Q \xrightarrow{\bar{a}b} Q'$ and $T' \sim_{ct} \llbracket Q' \rrbracket$. So we know

$$\llbracket P' \rrbracket \mid E[\langle Z \rangle(Z\langle b \rangle)] \approx_{ct} \llbracket Q' \rrbracket \mid E[\langle Z \rangle(Z\langle b \rangle)] \quad (1)$$

We want to show

$$P' \mathcal{R} Q' \quad \text{that is, } \llbracket P' \rrbracket \approx_{ct} \llbracket Q' \rrbracket \quad (2)$$

By setting E to be 0 in (1), we obtain (2), and thus close this case.

- $P \xrightarrow{\bar{a}(b)} P'$. By Lemma 6, $\llbracket P \rrbracket \xrightarrow{(b)\bar{a}[\langle Z \rangle(Z\langle b \rangle)]} T$ and $T \sim_{ct} \llbracket P' \rrbracket$. Since $\llbracket P \rrbracket \approx_{ct} \llbracket Q \rrbracket$, we know that $\llbracket P \rrbracket$ must be able to be matched by $\llbracket Q \rrbracket \xrightarrow{(b)\bar{a}[\langle Z \rangle(Z\langle b \rangle)]} T'$ (apply α -conversion if needed). This is because $\llbracket Q \rrbracket$ can only emit such form of processes, and moreover if the matching does not have a bound name (e.g., $\llbracket Q \rrbracket \xrightarrow{\bar{a}[\langle Z \rangle(Z\langle c \rangle)]} T''$) then one can design a context to distinguish $\llbracket P \rrbracket$ and $\llbracket Q \rrbracket$. So for every $E[X]$ s.t. $b \notin \text{fn}(E)$, we have $(b)(T \mid E[\langle Z \rangle(Z\langle b \rangle)]) \approx_{ct} (b)(T' \mid E[\langle Z \rangle(Z\langle b \rangle)])$. By Lemma 7, $Q \xrightarrow{\bar{a}(b)} Q'$ and $T' \sim_{ct} \llbracket Q' \rrbracket$. So we know

$$(b)(\llbracket P' \rrbracket \mid E[\langle Z \rangle(Z\langle b \rangle)]) \approx_{ct} (b)(\llbracket Q' \rrbracket \mid E[\langle Z \rangle(Z\langle b \rangle)]) \quad (3)$$

In terms of local bisimulation [5, 25], for every π process R , we need to show

$$(b)(P' \mid R) \mathcal{R} (b)(Q' \mid R) \quad \text{i.e., } (b)(\llbracket P' \rrbracket \mid \llbracket R \rrbracket) \approx_{ct} (b)(\llbracket Q' \rrbracket \mid \llbracket R \rrbracket) \quad (4)$$

Comparing equations (3) and (4), one can see that the different part is $E[\langle Z \rangle(Z\langle b \rangle)]$ and $\llbracket R \rrbracket$. Since the inverse of the encoding is a surjection, if all possible forms of E is iterated, $\llbracket R \rrbracket$ must be hit somewhere (i.e., some choice of E makes $E[\langle Z \rangle(Z\langle b \rangle)]$ and $\llbracket R \rrbracket$ equal). Therefore we infer that (4) is true and thus complete this case.

- $P \xRightarrow{\tau} P'$. By Lemma 6, $\llbracket P \rrbracket \xRightarrow{\tau} T$ and $T \sim_{ct} \llbracket P' \rrbracket$. Because $\llbracket P \rrbracket \approx_{ct} \llbracket Q \rrbracket$, we know $\llbracket Q \rrbracket \xRightarrow{\tau} T' \approx_{ct} T$. Then by Lemma 7, $Q \xRightarrow{\tau} Q'$ and $T' \sim_{ct} \llbracket Q' \rrbracket$. So we have $P' \mathcal{R} Q'$ because $\llbracket P' \rrbracket \sim_{ct} T \approx_{ct} T' \sim_{ct} \llbracket Q' \rrbracket$.

□

4 Normal bisimulation for $\Pi^{D,d}$

In this section, we show that context bisimulation in $\Pi^{D,d}$ can be characterized by the much simpler normal bisimulation.

The factorization theorem

Below is the factorization theorem in presence of parameterization on names (and on processes as well). We recall that \equiv is the structural congruence. As explained in Section 1, the upshot of establishing the factorization theorem is to find the right small processes so-called triggers. Here we have three kinds of triggers, to tackle different kinds of parameterizations. In particular, we stipulate that the triggers are as follows: $Tr_m^d \stackrel{\text{def}}{=} \langle z \rangle \bar{m}[\langle Y \rangle (Y \langle z \rangle)]$, $Tr_m^D \stackrel{\text{def}}{=} \langle Z \rangle \bar{m}Z$, and $Tr_m \stackrel{\text{def}}{=} \bar{m}$. These triggers are of somewhat a similar flavor but quite different in shape, with the aim at factorizing out respectively a name abstraction, a process abstraction and a non-abstraction process in certain context. The first trigger, i.e., Tr_m^d , is the main contribution of this work, whereas the other two are inherited from [26] and [19] respectively.

Theorem 10 (Factorization). *Given $E[X]$ of $\Pi^{D,d}$, it holds for every A , fresh m (i.e., $m \notin \text{fn}(E, A)$) that*

- (1) *if $E[X]$ is not an abstraction, then*
 - (i) *if A is not an abstraction, then $E[A] \approx_{ct} (m)(E[Tr_m] \mid !m.A)$;*
 - (ii) *if A is an abstraction on process, then $E[A] \approx_{ct} (m)(E[Tr_m^D] \mid !m(Z).A \langle Z \rangle)$;*
 - (iii) *if A is an abstraction on name, then $E[A] \approx_{ct} (m)(E[Tr_m^d] \mid !m(Z).Z \langle A \rangle)$.*
- (2) *else if $E[X]$ is an abstraction, i.e., $E[X] \equiv \langle \widetilde{U} \rangle E'$ for some non-abstraction E' (here $\langle \widetilde{U} \rangle$ denotes the abstractions prefixing E'), then*
 - (i) *if A is not an abstraction, then $E[A] \approx_{ct} \langle \widetilde{U} \rangle ((m)(E'[Tr_m] \mid !m.A))$;*
 - (ii) *if A is an abstraction on process, then $E[A] \approx_{ct} \langle \widetilde{U} \rangle ((m)(E'[Tr_m^D] \mid !m(Z).A \langle Z \rangle))$;*
 - (iii) *if A is an abstraction on name, then $E[A] \approx_{ct} \langle \widetilde{U} \rangle ((m)(E'[Tr_m^d] \mid !m(Z).Z \langle A \rangle))$.*

In Theorem 10, the clause (i) of (1) and (2) is actually Sangiorgi's seminal work [19]. The clause (ii) of (1) and (2) is analyzed in [26]. The clause (iii) of (1) and (2), which depicts the factorization for abstraction on names, can be discussed through a technical routine almost the same as (ii). With regard to more details we refer the reader to [19, 22, 26].

The method of *trigger* (including the technical approach) is well-developed in the field, due to the fundamental framework by Sangiorgi [22]. So the key to establishing factorization for processes allowing abstraction on names is the *trigger*, which is not known for a long time in contrast to the cases of abstraction on processes and that without abstractions. Once a right trigger is found, the rest of discussion is then almost standard. Below we give an example of the factorization concerning abstraction on names.

Example The basic idea of factorization concerning abstraction on names can be illustrated in the following example in which m is fresh (i.e., not in $A \langle d \rangle$).

$$A \langle d \rangle \approx_{ct} (m)(\langle \langle z \rangle \bar{m}[\langle Y \rangle (Y \langle z \rangle)] \rangle \langle d \rangle \mid m(Z).Z \langle A \rangle) \equiv (m)(\bar{m}[\langle Y \rangle (Y \langle d \rangle)] \mid m(Z).Z \langle A \rangle)$$

For example, if A is $\langle x \rangle \bar{x}b$, then $A\langle d \rangle \equiv \bar{d}b$, and

$$A\langle d \rangle \approx_{ct} (m)(\bar{m}[\langle Y \rangle(Y\langle d \rangle)] \mid m(Z).Z\langle A \rangle) \approx_{ct} (m)(\langle \langle Y \rangle(Y\langle d \rangle) \rangle \langle A \rangle) \equiv A\langle d \rangle \equiv \bar{d}b$$

Normal bisimulation for $\Pi^{D,d}$

Below is the definition of normal bisimulation whose clauses are designed with regard to the factorization theorem. We recall that $Tr_m \stackrel{\text{def}}{=} \bar{m}$, $Tr_m^D \stackrel{\text{def}}{=} \langle Z \rangle \bar{m}Z$, and $Tr_m^d \stackrel{\text{def}}{=} \langle z \rangle \bar{m}[\langle Y \rangle(Y\langle z \rangle)]$.

Definition 11. A symmetric binary relation \mathcal{R} on closed processes of $\Pi^{D,d}$ is a normal bisimulation, if whenever $P \mathcal{R} Q$ the following properties hold:

1. If $P \xrightarrow{a(Tr_m)} P'$ (m is fresh w.r.t. P and Q), then $Q \xrightarrow{a(Tr_m)} Q'$ for some Q' s.t. $P' \mathcal{R} Q'$;
2. If $P \xrightarrow{a(Tr_m^D)} P'$ (m is fresh w.r.t. P and Q), then $Q \xrightarrow{a(Tr_m^D)} Q'$ for some Q' s.t. $P' \mathcal{R} Q'$;
3. If $P \xrightarrow{a(Tr_m^d)} P'$ (m is fresh w.r.t. P and Q), then $Q \xrightarrow{a(Tr_m^d)} Q'$ for some Q' s.t. $P' \mathcal{R} Q'$;
4. If $P \xrightarrow{(\tilde{c})\bar{a}A} P'$ and A is not an abstraction, then $Q \xrightarrow{(\tilde{d})\bar{a}B} Q'$ for some \tilde{d}, Q' and B that is not an abstraction, and it holds that (m is fresh) $(\tilde{c})(P' \mid !m.A) \mathcal{R} (\tilde{d})(Q' \mid !m.B)$.
5. If $P \xrightarrow{(\tilde{c})\bar{a}A} P'$ and A is an abstraction on process, then $Q \xrightarrow{(\tilde{d})\bar{a}B} Q'$ for some \tilde{d}, Q' and B that is an abstraction on process, and it holds that (m is fresh) $(\tilde{c})(P' \mid !m(Z).A\langle Z \rangle) \mathcal{R} (\tilde{d})(Q' \mid !m(Z).B\langle Z \rangle)$.
6. If $P \xrightarrow{(\tilde{c})\bar{a}A} P'$ and A is an abstraction on name, then $Q \xrightarrow{(\tilde{d})\bar{a}B} Q'$ for some \tilde{d}, Q' and B that is an abstraction on name, and it holds that (m is fresh) $(\tilde{c})(P' \mid !m(Z).Z\langle A \rangle) \mathcal{R} (\tilde{d})(Q' \mid !m(Z).Z\langle B \rangle)$.
7. If $P \xrightarrow{\tau} P'$, then $Q \xrightarrow{} Q'$ for some Q' s.t. $P' \mathcal{R} Q'$;

Process P is normal bisimilar to Q , written $P \approx_{nr} Q$, if $P \mathcal{R} Q$ for some normal bisimulation \mathcal{R} . Relation \approx_{nr} is called normal bisimilarity, and is a congruence (see [19] for a reference). The strong version of \approx_{nr} is denoted by \sim_{nr} .

Coincidence between normal bisimilarity and context bisimilarity in $\Pi^{D,d}$

Now we have the following theorem. The detailed proof is referred to [27].

Theorem 12. In $\Pi^{D,d}$, normal bisimilarity coincides with context bisimilarity; that is, $\approx_{nr} = \approx_{ct}$.

5 Conclusion

In this paper, we have exhibited a new encoding of name-passing in the higher-order paradigm that allows parameterization, and a normal bisimulation in that setting as well. In the former, we demonstrate the conformance of the encoding to the well-established criteria in the literature. In the latter, we prove the coincidence between normal and context bisimulation by pinpointing how to factorize an abstraction on some name. The encoding of this work is inspired by the one proposed by Alan Schmitt during the communication concerning another work. That encoding, as given below, somewhat swaps the roles of input and output and treats $a(x).P$ somehow as $a.\langle x \rangle P$ (like those calculi admitting abstractions and concretions [19]).

$$\begin{aligned} \llbracket a(x).P \rrbracket &\stackrel{\text{def}}{=} \bar{a}[\langle x \rangle \llbracket P \rrbracket] \\ \llbracket \bar{a}b.Q \rrbracket &\stackrel{\text{def}}{=} a(Y).(Y\langle b \rangle \mid \llbracket Q \rrbracket) \end{aligned}$$

From the angle of achieving first-order interaction, the encoding strategy above is truly interesting. However, it appears not to satisfy some usual operational correspondence (say, in [8] or [12]), and full abstraction is not quite clear. Based on the results in this paper, it is tempting to expect that this encoding have some (nearly) same properties, and this is worthwhile for more investigation.

The results of this paper can be dedicated to facilitate further study on the expressiveness of higher-order processes. The following questions, among others, are still open: whether π can be encoded in a higher-order setting only allowing parameterization on processes; whether there is a better encoding of π than the one in [28], using higher-order processes only capable of parameterization on names; whether Π^d afford a normal-like characterization of context bisimulation.

Acknowledgements We thank the anonymous referees for their useful comments on this article.

References

- [1] F. Bonchi, D. Petrisan, D. Pous & J. Rot (2015): *Lax Bialgebras and Up-To Techniques for Weak Bisimulations*. In: *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR 2015)*, *Leibniz International Proceedings in Informatics (LIPICS)* 42, pp. 240–253, doi:10.4230/LIPIcs.CONCUR.2015.240.
- [2] M. Bundgaard, T. Hildebrandt & J. C. Godskesen (2006): *A CPS Encoding of Name-passing in Higher-order Mobile Embedded Resources*. *Theoretical Computer Science* 356(3), pp. 422–439, doi:10.1016/j.tcs.2006.02.006.
- [3] U. H. Engberg & M. Nielsen (1986): *A Calculus of Communicating Systems with Label Passing*. Technical Report DAIMI PB-208, Computer Science Department, University of Aarhus. Available at <http://www.daimi.au.dk/PB/208/>.
- [4] U. H. Engberg & M. Nielsen (2000): *A Calculus of Communicating Systems with Label Passing - Ten Years After*. In: *Proof, Language, and Interaction: Essays in Honour of Robin Milner*, MIT Press Cambridge, pp. 599–622.
- [5] Yuxi Fu (2005): *On Quasi Open Bisimulation*. *Theoretical Computer Science* 338(1-3), pp. 96–126, doi:10.1016/j.tcs.2004.10.041.
- [6] Yuxi Fu (2015): *Theory of interaction*. *Theoretical Computer Science*, doi:10.1016/j.tcs.2015.07.043.
- [7] Yuxi Fu & Hao Lu (2010): *On the Expressiveness of Interaction*. *Theoretical Computer Science* 411, pp. 1387–1451, doi:10.1016/j.tcs.2009.11.011.
- [8] D. Gorla (2008): *Towards a Unified Approach to Encodability and Separation Results for Process Calculi*. In: *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR 2008)*, LNCS 5201, Springer Verlag, pp. 492–507, doi:10.1007/978-3-540-85361-9_38.
- [9] D. Gorla & U. Nestmann (2016): *Full Abstraction for Expressiveness: History, Myths and Facts*. *Mathematical Structures in Computer Science* 26, pp. 639–654, doi:10.1017/S0960129514000279.
- [10] Daniele Gorla (2009): *On the Relative Expressive Power of Calculi for Mobility*. *Electronic Notes in Theoretical Computer Science* 249, pp. 269–286, doi:10.1016/j.entcs.2009.07.094.
- [11] D. Kouzapas, J. A. Pérez & Nobuko Yoshida (2016): *On the Relative Expressiveness of Higher-Order Session Processes*. In: *Proceedings of the 25th European Symposium on Programming (ESOP 2016)*, LNCS, pp. 446–475, doi:10.1007/978-3-662-49498-1_18.
- [12] I. Lanese, J. A. Pérez, D. Sangiorgi & A. Schmitt (2010): *On the Expressiveness of Polyadic and Synchronous Communication in Higher-Order Process Calculi*. In: *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2010)*, LNCS, Springer Verlag, pp. 442–453, doi:10.1007/978-3-642-14162-1_37.

- [13] I. Lanese, J.A. Pérez, D. Sangiorgi & A. Schmitt (2008): *On the Expressiveness and Decidability of Higher-Order Process Calculi*. In: *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LICS 2008)*, IEEE Computer Society, pp. 145–155, doi:10.1109/LICS.2008.8. Journal version in [?].
- [14] S. Lenglet, A. Schmitt & J.-B. Stefani (2009): *Normal Bisimulations in Calculi with Passivation*. In: *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FOSSACS 2009)*, LNCS 5504, Springer Verlag, pp. 257–271, doi:10.1007/978-3-642-00596-1_19.
- [15] S. Lenglet, A. Schmitt & J.-B. Stefani (2011): *Characterizing Contextual Equivalence in Calculi with Passivation*. *Information and Computation* 209, pp. 1390–1433, doi:10.1016/j.ic.2011.08.002.
- [16] R. Milner (1989): *Communication and Concurrency*. Prentice Hall.
- [17] R. Milner, J. Parrow & D. Walker (1992): *A Calculus of Mobile Processes (Parts I and II)*. *Information and Computation* 100(1), pp. 1–77, doi:10.1016/0890-5401(92)90008-4, 10.1016/0890-5401(92)90009-5.
- [18] J. Parrow (2016): *General Conditions for Full Abstraction*. *Mathematical Structures in Computer Science* 26, pp. 655–657, doi:10.1017/s0960129514000280.
- [19] D. Sangiorgi (1992): *Expressing Mobility in Process Algebras: First-order and Higher-order Paradigms*. Phd thesis, University of Edinburgh.
- [20] D. Sangiorgi (1996): *Bisimulation for Higher-order Process Calculi*. *Information and Computation* 131(2), pp. 141–178, doi:10.1006/inco.1996.0096.
- [21] D. Sangiorgi (1998): *On the Bisimulation Proof Method*. *Mathematical Structures in Computer Science* 8(6), pp. 447–479, doi:10.1017/S0960129598002527.
- [22] D. Sangiorgi & D. Walker (2001): *The Pi-calculus: a Theory of Mobile Processes*. Cambridge University Press.
- [23] B. Thomsen (1990): *Calculi for Higher Order Communicating Systems*. Phd thesis, Department of Computing, Imperial College.
- [24] B. Thomsen (1993): *Plain CHOCS, a Second Generation Calculus for Higher-Order Processes*. *Acta Informatica* 30(1), pp. 1–59, doi:10.1007/BF01200262.
- [25] Xian Xu (2012): *Distinguishing and Relating Higher-order and First-order Processes by Expressiveness*. *Acta Informatica* 49(7-8), pp. 445–484, doi:10.1007/s00236-012-0168-9.
- [26] Xian Xu (2013): *On Context Bisimulation for Parameterized Higher-order Processes*. In: *Proceedings of the 6th Interaction and Concurrency Experience (ICE 2013)*, EPTCS 131, pp. 37–51, doi:10.4204/EPTCS.131.5.
- [27] Xian Xu (2016): *Higher-order Processes with Parameterization over Names and Processes (with appendices)*. Available at <http://basics.sjtu.edu.cn/~xuxian/express2016withappendices.pdf>.
- [28] Xian Xu, Qiang Yin & Huan Long (2015): *On the Computation Power of Name Parameterization in Higher-order Processes*. In: *Proceedings of 8th Interaction and Concurrency Experience (ICE 2015)*, EPTCS 189, pp. 114–127, doi:10.4204/EPTCS.189.10.